
ANSI/NIST ITL 2011 Type 98 Best Practice Implementation Guidance

For the Assurance of Biometric Data Integrity, Authenticity and
Auditable Chain of Custod

Version 1.3
June 24th, 2011

Revision History

Version	Date	Summary of changes
1.0	14-Jan-2011	Initial Draft
1.1	14-April-2011	Updated to Reflect ANSI/NIST ITL 2011 Draft 3
1.2	24-May-2011	Updated to Reflect ANSI/NIST ITL 2011 Draft 4
1.3	24-June-2011	Updated to Reflect CMS SME Comments

TABLE OF CONTENTS

1.	Introduction	1
1.1	Best Practices Assumptions and Requirements	1
1.2	Maximizing ITL Interoperability	1
2.	Best Practice Population of Type 98 Parent Fields	2
2.1	IA Data Format Owner 98.003	2
2.2	IA Data Format Type 98.005	2
2.3	IA Audit Log Field 98.900	2
2.4	IA Audit Revision Number 98.901	2
2.4.1	Creating the Audit Log	3
2.4.2	Processing the Audit Log	3
2.4.2.1	Securing the Audit Log	3
2.4.2.2	Audit Reporting Methodology	4
2.4.2.3	Reconstruction Methodology	4
2.4.2.4	Audit Log Field Extraction	5
3.	Best Practice Population of Type 98 User-Defined Security Fields	6
3.1	XML Version	6
3.1.1	XML BI Creation Guidelines	7
3.1.2	XML BI Processing\Validation Guidelines	8
3.2	Traditional Encoding Version:	9
3.2.1	Traditional Encoding BI Creation Guidelines	10
3.2.2	Traditional BI Validation Guidelines	11
	Appendix A Acronyms	12
	Appendix B Sample XML AuditLog Field	13
	Appendix C Sample BI within XML Type 98	14
	Appendix D Binding Information Specification (For Traditional Encoding)	18
	Appendix E References	34

FIGURES

Figure 1: Sample XML Type 98 Audit Log Field.....	3
Figure 3: XML Audit Log Reference in Manifest.....	4
Figure 4: Sample Audit Statement.....	4
Figure 5: File Version Reconstruction.....	5

TABLES

Table 1: Best Practice XML Type 98	7
Table 2: Best Practice Traditional Type 98	10

DRAFT

1. INTRODUCTION

This document describes the best practice for implementation of the ANSI/NIST ITL-2011 Type 98 logical record. Best practices assure the integrity and authenticity of biometric modalities and audit custody chains.

The ITL Information Assurance (IA) logical record (Type 98) contains user-defined security information. While child standards are given the flexibility to use different mechanisms to implement the Type 98 record, best practices include the use of Cryptographic Binding (CB) techniques within the Type 98 IA record. The benefits of using the recommended CB methodology in conjunction with the audit field include improved interoperability, assurance of integrity over a file and composite logical records, aggregation of modalities, granular detection of modifications, and assured file versioning.

Cryptographic Binding is a methodology for providing integrity and authenticity to data and data relationships using well-known cryptographic techniques. CB provides the ability to detect granular modifications, insertions, deletions, or unauthorized data sources and facilitates assured synchronization of data and data collections. CB is also leveraged in combination with the Audit Log Field (ALF) to assure chain of custody and allow for automated reconstruction of previous versions.

This document gives detailed guidelines for defining, populating, and processing the ITL Type 98 and provides sample user-defined fields using the CB technique. Guidance is provided for both XML and traditional encoding schemes.

1.1 Best Practices Assumptions and Requirements

- Access to Cryptographic software libraries for hashing, digital signatures, and certificate validation
- Participation in an existing Public Key Infrastructure (PKI)
- Consistent parsing of ITL files
- Canonicalization of file formats to ensure unbroken signatures

1.2 Maximizing ITL Interoperability

To maximize interoperability on multiple domains that may not have access to the same Key Management Infrastructures, the Type 98 best practice includes the following features:

- *Asymmetric cryptography.* Asymmetric cryptography allows users with access to a public key and CA to validate integrity and authenticity, without requiring a shared secret.
- *Multiple instances.* The existence of multiple instances of Type 98 allows systems to use the instances they understand and ignore the instances that cannot be validated in their respective domains.
- *Assured versioning.* The ALF allows users from one domain to reconstruct and validate previous versions of a record.

2. BEST PRACTICE POPULATION OF TYPE 98 PARENT FIELDS

The ANSI/NIST ITL 2011 standard describes a general Type 98 record. The type exists to provide a consistent location for IA data pertaining to the ITL file. The Type 98 record contains several fields common across all record types (e.g. IDC, SRC). Refer to [ANSI/NIST ITL 2011] for details on these common fields. The following subsections describe unique fields specified by the ITL 2011 standard for Type 98 records, and guidance for population of these fields.

2.1 IA Data Format Owner 98.003

The IA data generated by the CB method is called **Binding Information (BI)**. The format owner for the CB BI implementation of Type 98 is NSA's Secure Data Enabling Technologies. This authority is assigned a four digit hex value by the IBIA.

2.2 IA Data Format Type 98.005

BI can be encoded using either an XML or binary format type. Both types are assigned a four digit hex value by the IBIA. The combination of Format Owner and Format Type uniquely identify the BI format.

2.3 IA Audit Log Field 98.900

The ALF (98.900) consists of a series of change statements. Each change statement describes a discrete change made to a referenced logical record since the previous Type 98 was created (note that this does not include the Type 98 record itself). Best practices dictate that every change made to an ITL record **SHOULD** be logged in the Type 98 ALF in order to preserve chain of custody and support reconstruction of previous versions. Moreover, if a given entity wishes to construct multiple Type 98s for different target domains, each Type 98 must have identical ALFs with identical revision numbers. An ALFs must be bound by the cryptographic binding (see Figure 1 for a Sample Audit Log Field).

The Audit Log Field consists of a collection of fields and subfields as defined by the ANSI/NIST ITL 2011 Standard.

2.4 IA Audit Revision Number 98.901

Additionally, each Type 98 needs to be marked with a unique audit log revision number (ARN) to support identification of previous file versions. This field must be as defined by the ANSI/NIST ITL 2011 standard.

2.4.1 Creating the Audit Log

The Audit log consists of a series of statements for each individual change made to other logical records since the previous Type 98 was created, such as a change in biographic data or an addition of a new record. A Type 98 record with an audit log should be created every time an agent saves changes to an ITL file.

Software used to modify an ITL must generate an event for each modification to the ITL, which will be used to populate the ITL audit log on save. See Figure 1 for a sample audit log field.

```
<!-- 98.900 IA Audit Log Field (ALF) -->
<!-- Events should be added in the order they occurred so they can be reversed successfully -->
<AuditLog id="98.900">
  <Event Field="1.003" Type="Modification" EventReason="Additional Images were acquired" Agent="zsimonetti">
    <OldValue <!-- Old value of the attribute goes here (if the field is an attribute) -->>
      <!-- Old value of the element goes here (if the field is an element)-->
    </OldValue>
  </Event>
</AuditLog>
<!-- 98.901 IA Audit Revision Number (ARN) -->
<AuditRevisionNumber id="98.901" Revision="1">
</AuditRevisionNumber>
```

Figure 1: Sample XML Type 98 Audit Log Field

2.4.2 Processing the Audit Log

The Type 98 Audit Log can be leveraged both to generate audit reports and automate reconstruction of previous ITL versions. To protect the audit log data, the audit field must be included within the protection scheme (e.g. cryptographic binding) applied to all other records protected by the Type 98. This section describes best practices for securing the audit log, auto-generating previous versions, and producing audit reports for an encoding agnostic ITL file. The audit log may be omitted in cases where it becomes a performance problem, but this will break the chain of custody. It is also possible to extract the audit log and store it separately if necessary.

2.4.2.1 Securing the Audit Log

The audit field must be included within the cryptographic binding applied to all other records protected by the Type 98. For example, in XML, store a hash of the audit log in the XML Digital Signature (DSig) Manifest. The manifest reference ID is created by concatenating the UDC with the URI. The URI and type reference attributes may also be optionally populated. Each reference also identifies the hash algorithm used (DigestMethod), and the hash value (Digest Value). Figure 3 illustrates this example:

```

<Manifest>
  <!--References to other logical records here-->
  <!-- Manifest Reference to the Audit log for the file -->
  <Reference Id="500#98.900" URI="#98.900" type="AuditLog">
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
    <DigestValue>60NvZvtdTB+7UnlLp/H24p7h4bs=</DigestValue>
  </Reference>
</Manifest>

```

Figure 2: XML Audit Log Reference in Manifest

2.4.2.2 Audit Reporting Methodology

The Audit Log provides secure audit data for ITL records and files by recording discrete changes made to logical records and digital signatures to identify the agent of change. The audit log can be parsed to and filtered to generate different types of audit reports. For example, if a report was needed on changes made by a specific agent, events with the agent attribute matching the target agent could be extracted and reported. Reports could also be generated based on changes made to only to specific logical records, or different types of events, or for specific event reasons. The XML block below shows a sample event element, with the attributes that could be leveraged for different types of audit reports.

```

<!-- 98.900 IA Audit Log Field (ALF) -->
<!-- Events should be added in the order they occurred so they can be reversed successfully -->
<AuditLog id="98.900">
  <Event Field="1.003" Type="Modification" EventReason="Additional Images were acquired" Agent="zsimonetti">
    <OldValue <!-- Old value of the attribute goes here (if the field is an attribute) -->>
      <!-- Old value of the element goes here (if the field is an element)-->
    </OldValue>
  </Event>
</AuditLog>
<!-- 98.901 IA Audit Revision Number (ARN) -->
<AuditRevisionNumber id="98.901" Revision="1">
</AuditRevisionNumber>

```

Figure 3: Sample Audit Statement

To audit changes made to the ITL file over time or in a given time frame, the Type 98 Data Creation Date (98.006) should be leveraged.

2.4.2.3 Reconstruction Methodology

The Audit Log can also be leveraged for automated reconstruction with cryptographic validation. Given an unbroken line of Type 98's with accurate audit logs and BIs, any previous version of the ITL file can be reconstructed and cryptographically validated. The accuracy of this reconstruction can be verified by comparing hashes of the reconstructed version to the hashes in the original Type 98 BI. The figure below depicts the process for automated reconstruction.

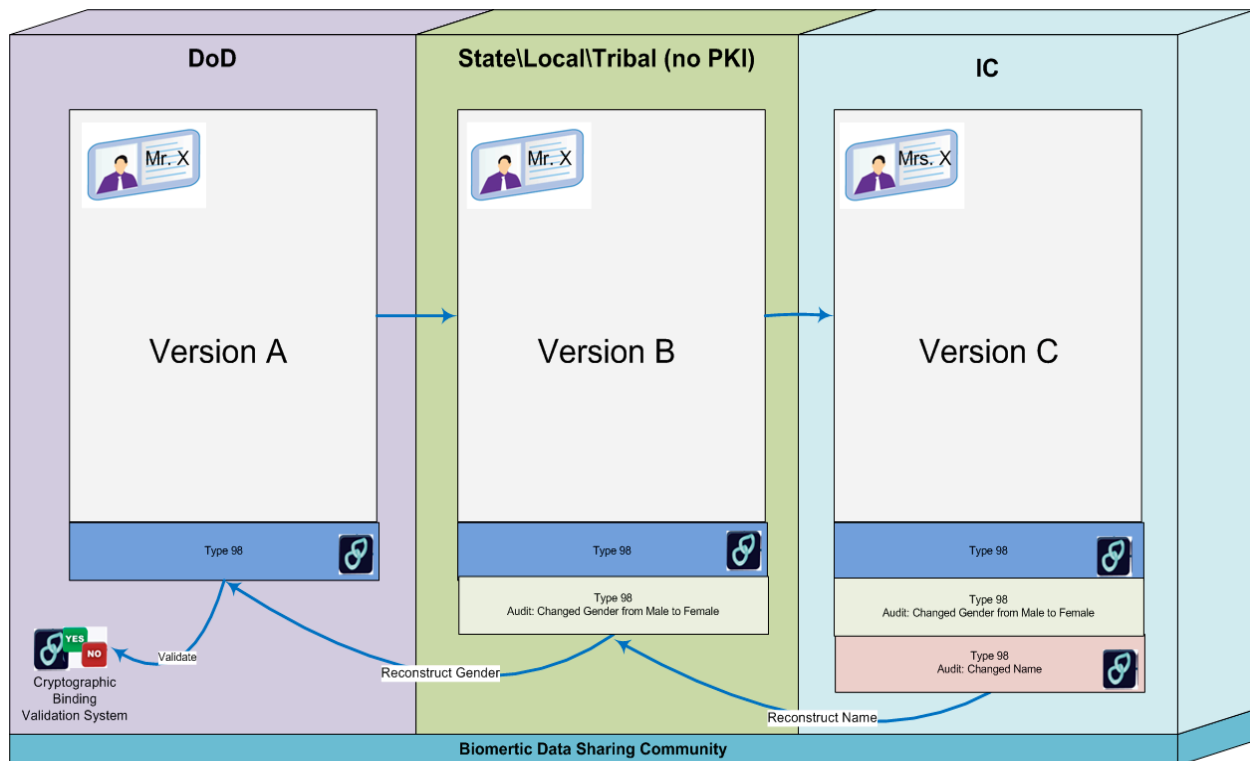


Figure 4: File Version Reconstruction

Steps to automate reconstruction of previous versions of the ITL:

- Start with the current ITL (version n), and create a temporary empty ITL file ITL (version n-1).
- Parse the latest Type 98 audit log field to obtain the Old Value(s) for the modified record(s) described within the audit field.
- Populate Version n-1 using the version n values, and then replace modified fields with the Old Value(s) specified in the latest Type 98.
- Hash each logical record within version n-1, and compare these to the hashes stored within the previous Type 98/CB.
- Continue recursively reconstructing versions from the remaining Type 98 records (ordered by IDC) until you come to desired version.

2.4.2.4 Audit Log Field Extraction

In cases where the audit log becomes too large for transport as part of the ITL, users may consider parsing and removing the ALF and storing it separately from the ITL file with appropriate metadata tags.

3. BEST PRACTICE POPULATION OF TYPE 98 USER-DEFINED SECURITY FIELDS

In addition to the general fields described in section 2, the Type 98 best practice relies on user-defined fields (98.200-899) for storing BI data. While child standards are expected to utilize these fields to define the particular IA mechanism or format which best suits their needs (e.g. signature generated using XML DSig), best practices recommend that child standards use “Binding Information” (BI) created by the CB process. The BI creation procedure, format, and field location within the Type 98 user defined fields varies based on the encoding scheme. This section describes best practices for creation and processing of BI for XML and Traditional encoding schemas.

3.1 XML Version

When using XML encoding, the BI consists of XML Digital Signature with a manifest and signature block. The manifest is used to allow for granularity in validation and verification¹. By placing references that do not explicitly pertain to the Type 98 record (i.e. references to the other logical records) in the manifest it is possible to have the signature validate even if a subset of logical records have been modified. This allows “subset validation,” assuring that subsets of logical records ITL file have not been altered and that the hashes of the referenced data objects (e.g. logical records) have not been altered.

The Manifest (98.200) and Signature (98.201) user-defined fields are used when creating a BI within the Type 98 using the XML ITL encoding as illustrated by Table 1:

Mnemonic	Cond code	Field number	Field name	Occur count	
				Min	Max
	Mandatory	98.001	Record Header	1	1
IDC	Mandatory	98.002	Image Designation Character	1	1
DFO		98.003	IA Data Format Owner	1	1
ORG		98.004	Originating Agency	1	1
ORI			Originating Agency Identifier	1	1

¹ In XML Dsig, when references are contained within the signed data element any references that cannot be found because they have been removed from the XML document will break the signature. Utilizing a manifest however allows the signature to remain valid even when the reference to a logical record is removed from the document, thus allowing the remaining records to be trusted.

OAN			Originating Agency Name	0	1
DFT	Mandatory	98.005	IA Data Format Type	1	1
DCD	Mandatory	98.006	IA Data Creation Date	1	1
RSV		98.007-98.199	Reserved For Future Definition		
MF	Mandatory	98.200	Manifest	1	1
CB	Mandatory	98.201	Signature	1	1
UDF		98.202-98.899	User-Defined Fields		
ALF	Mandatory	98.900	IA Audit Log Field	1	1
ARN	Mandatory	98.901	Audit Revision Number	1	1
RSV		98.901-995	Reserved For Future Definition		

Table 1: Best Practice XML Type 98

3.1.1 XML BI Creation Guidelines

The goal of the CB is to assure that no modalities have been tampered with, and that their relation to one another has integrity and authenticity (e.g. to prove that a set of fingerprints belongs to the same person as a set of iris scans and that they come from a trusted source).

A Type 98 should be created on save, each time an ITL file is updated by an agent. To populate the XML Type 98, logical records are parsed from the ITL, canonicalized, hashed, and used to populate the XML Manifest and Signature. On receiving an ITL, the Type 98 is validated through a similar process.

Processing

To create a BI within an XML ITL file, each logical record must be consistently processed in order to maintain cryptographic interoperability of the hash values. Each logical record corresponds to an XML element which must be identified, put into a unique, canonicalized form and converted to bytes in order to be hashed.

In addition to canonicalization, the system must determine a unique identifier for each object intended to be bound within the CB. NIST recommends the use of the IDC for each logical record. Note that some logical records have non-unique IDCs (e.g. fingerprint minutiae). In cases such as this all records with the same IDC should be treated as a single object, and bound accordingly (as the fingerprint data is useless without the minutiae data).

In order to ensure consistency of the hashes for records with identical IDCs, records with the same IDC should be aggregated into a canonical form prior to hashing. NIST recommends constructing this canonical form by appending the logical records in order of increasing record type (e.g. append Type_17 to Type_13 to Type_7).

Field Population

The Manifest field (98.200) should contain a reference to each logical record, the ALF (98.900) and ARN (98.901) of the current record, and the IA Data Creation Date (98.006). These objects should be parsed and hashed and the hash stored in the reference along with the algorithm used to generate the hash. The resource should also have an ID attribute set to give a unique name to the resource. The Image Designation Character is the recommended ID attribute. Though not required, it is suggested that URI and Type attributes be used as well. The URI to the referenced resource and its type can assist automated validations. The following is an example of a reference:

```
<Reference Id="500#98.900" URI="#98.900" Type="AuditLogField">
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>60NvZvtdTB+7UnLp/H24p7h4bs=</DigestValue>
</Reference>
```

The ID attribute should contain the IDC number for the asset. For the ALF it should be the Image Designation Character with ‘#98.900’ appended.

The signature block (98.201) should include a reference with a hash to the BindingAttributes, Image Designation Character (98.001), IA Data Creation Date (98.006), and the Manifest (98.200) inside a SignedInfo element. Each logical record must have a unique canonicalization transform, which (in addition to standard XML canonicalization) sets a fixed structure for the order of all attributes and child elements within the logical record structure in order to ensure hash digest consistency. The SignedInfo field should also contain information about the canonicalization method and SignatureMethod used. The SignedInfo should be canonicalized and the signature method should be applied over the SignedInfo field and stored in the SignatureValue field. The relevant KeyInfo should also be stored as shown in Appendix C.

3.1.2 XML BI Processing\Validation Guidelines

To validate the Type 98 record, the signature must first be verified followed by validation of the individual references within the manifest (as needed).

The validation of the signature will ensure that the BindingAttributes, Image Designation Character (98.001), IA Data Creation Date (98.006), and the Manifest (98.200) have not changed. However, the items referenced in the manifest must be validated separately as the signature only validates that the hash values were not changed.

After the signature is validated the references in the manifest should be hashed using the algorithm specified in the manifest reference. Once generated, hashes are compared to the hashes contained in the manifest. If the hashes match then the asset has integrity.

3.2 Traditional Encoding Version:

When using the Traditional ITL encoding, the BI consists of an ASN.1 Cryptographic Message Syntax object called a ContentCollection (See: RFC 4073) which is embedded within field 98.200.

To create BI within a Traditional ITL Type 98, use the following user-defined fields:

Mnemonic	Cond code	Field number	Character Type	Field name	Occur count	
					Min	Max
	Mandatory	98.001		Record Header	1	1
IDC	Mandatory	98.002	Numeric	Image Designation Character	1	1
DFO		98.003	AlphaNumeric	IADB Format Owner	1	1
ORG		98.004	AlphaNumeric	Originating Agency	1	1
ORI			Originating Agency Identifier	1	1	ORI
OAN			Originating Agency Name	0	1	OAN
DFT	Mandatory	98.005	AlphaNumeric	IADB Format Type	1	1
DCD	Mandatory	98.006	Numeric	IA Data Creation Date	1	1
RSV		98.007-98.199		Reserved For Future Definition		
CB	Mandatory	98.200	Binary (ASN.1)	Binding Information	1	1

UDF		98.201-98.899		User-Defined Fields		
ALF	Mandatory	98.900	AlphaNumeric	IA Audit Log	1	1
ARN	Mandatory	98.901	Audit Revision Number	1	1	ARN
RSV		98.901-995		Reserved For Future Definition		

Table 2: Best Practice Traditional Type 98

3.2.1 Traditional Encoding BI Creation Guidelines

The goal of the CB is to assure that no modalities have been tampered with, and that their relation to one another has integrity and authenticity (e.g. to prove that a set of fingerprints belongs to the same person as a set of iris scans).

A Type 98 should be created on save, each time an ITL file is updated by an agent. To populate the Traditional Type 98, logical records are parsed from the ITL, canonicalized, hashed, and used to populate the BI object. On receiving an ITL, the Type 98 is validated through a similar process.

Parsing

To create a CB BI within a Traditional ITL file, each logical record must be consistently parsed in order to maintain cryptographic interoperability of the hash values. Consistent parsing of records is beginning with the first bytes of an ITL file, which always correspond to the Type 1 record. The first field of every type indicates the length of that record. The records can then be individually parsed out recursively according to the following pseudo-code:

for (i=1; i<=99; i++)

- *Locate field 1 of record type I;*
- *Determine length type I;*
- *Parse out type I;*
- *Pass parsed bytes to hashing method;*

In addition to consistent parsing, a system must determine a unique identifier for each object intended to be bound within the CB. NIST recommends the use of the IDC for each logical record. Note that some logical records have non-unique IDCs (e.g. fingerprint minutiae). In cases such as this all records with the same IDC should be treated as a single object, and bound accordingly (as the fingerprint data is useless without the minutiae data).

In order to ensure consistency of the hashes for records with identical IDCs, records with the same IDC should be aggregated into a canonical form prior to hashing. NIST recommends constructing this canonical form by appending the logical records in order of increasing record type (e.g. append Type_17 to Type_13 to Type_7).

Field Population

Once all hashes have been created, Binding Information (See Appendix **Error! Reference source not found.**) must be generated. Once the Binding Information is generated, it must be inserted into field 98.200.

The Binding Information protects the integrity and authenticity of the data assets (i.e. biometric modalities) using a single ContentCollection object (as defined by RFC 4073). The IDCs should be used as “BindingDataReference” objects as defined by Appendix D. The BindingDataReference objects allow the hashes to be uniquely mapped to the corresponding data objects (i.e. logical records), and disambiguate each data asset’s hash from the hash collection.

3.2.2 Traditional BI Validation Guidelines

In order to process a Traditional ITL file containing a Type 98 with BI, the system must first locate the binding information and verify the digital signature. After which the system must parse each logical record in the same manner as described in section 3.2.1 and compare these hashes to the hashes stored within the BI’s ContentCollection.

Appendix A Acronyms

BI	Binding Information
CB	Cryptographic Binding
RI	Reference Implementation
IDC	Image Designation Character
SRC	Source Agency
DFO	Data Format Owner
DFT	Data Format Type
DCD	Data Creation Date
RSV	Reserved
MF	Manifest
UDF	User-Defined Field
ALF	Audit Log Field
RSV	Reserved

Appendix B Sample XML Audit Log Field

```
<!-- 98.900 IA Audit Log Field (ALF) -->
<!-- Events should be added in the order they occurred so they can be reversed successfully -->
<AuditLog id="98.900">
  <Event Field="1.003" Type="Modification" EventReason="Additional Images were acquired" Agent="zsimonetti">
    <OldValue <!-- Old value of the attribute goes here (if the field is an attribute) -->>
      <!-- Old value of the element goes here (if the field is an element)-->
    </OldValue>
  </Event>
</AuditLog>
<!-- 98.901 IA Audit Revision Number (ARN) -->
<AuditRevisionNumber id="98.901" Revision="1">
</AuditRevisionNumber>
```

Appendix C Sample BI within XML Type 98

```
<?xml version="1.0" encoding="utf-8">
<InformationAssuranceRecord
  xmlns:ansi-nist="http://niem.gov/niem/ansi-nist/2.0"
  xmlns:itl="http://biometrics.nist.gov/standard/2-2008"
  xmlns:nc="http://niem.gov/niem/niem-core/2.0"
  xmlns="http://samplenamespace.com/schemas/cryptogrtaphicbinding/type98"

- <InformationAssuranceRecord>
  <!-- 98.001 Record Type -->
  <ansi-nist:RecordCategoryCode id="98.001">98</ansi-nist:RecordCategoryCode>

  <!-- 98.002 Image Designation Character (IDC) -->
- <ansi-nist:ImageReferenceIdentification id="98.002">
  <nc:IdentificationID>500</nc:IdentificationID>
  </ansi-nist:ImageReferenceIdentification>

  <!-- 98.003 Source Agency/ORI (SRC) -->
  <ansi-nist:Organization id="98.003">
    <nc:OrganizationIdentification>
      <nc:IdentificationID>DAI000000</nc:IdentificationID>
    </nc:OrganizationIdentification>
  </ansi-nist:Organization>

  <!-- 98.004 Format Owner -->
  <ansi-nist:Organization id="98.004">
    <nc:OrganizationIdentification>
      <nc:IdentificationID>MDNISTIMG</nc:IdentificationID>
    </nc:OrganizationIdentification>
  </ansi-nist:Organization>

  <!-- 98.005 Format Type -->
  <Format id="98.005">Granular Cryptographic Binding</Format>

  <!-- 98.006 IA Data Creation Date (DCD) -->
  <ansi-nist:TransactionUTCDate Id="98.006">
    <nc:DateTime>2009-09-21T15:27:43Z</nc:DateTime>
  </ansi-nist:TransactionUTCDate>

  <!-- 98.200-899 User-Defined Fields -->
- <itl:ExampleUserDefinedFields>
- <!-- Well-formed XML goes here. Users may define a substitute element. -->

  <!-- This is the list of all the logical records and their digest and the hash
algorithm used
to create the digest. -->
  <Manifest Id="98.200">
    <!-- Reference to the Type 1 for the file -->
    <Reference Id="1">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>60NvZvt dB+7UnLLp/H24p7h4bs</DigestValue>
    </Reference>

    <!-- Reference to the Type 2 for the file -->
    <Reference Id="2">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>60NvZvt dB+7UnLLp/H24p7h4bs</DigestValue>
```

```

    </Reference>
  </Manifest>

  <!-- Example of a Cryptographic Binding in the user defined fields. -->
  <Signature Id="98.201" xmlns="http://www.w3.org/2000/09/xmldsig#" >
    <SignedInfo>
      <CanonicalizationMethod Algorithm=
"http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />

      <!-- Reference to binding attributes (i.e. Version, Method, Binder, Requestor,
Security Markings) -->
      <Reference URI="#BindingAttributes" Type=
"http://www.w3.org/2000/09/xmldsig#SignatureProperties">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>60NvZvt dTB+7UnLLp/H24p7h4bs=</DigestValue>
      </Reference>

      <!-- Reference to the IA Creation Date (i.e. the logical records) -->
      <Reference URI="#98.005" Type="ansi-nist:TransactionUTCDate">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>60NvZvt dTB+7UnLLp/H24p7h4bs=</DigestValue>
      </Reference>

      <!-- Reference to the manifest of items being bound (i.e. the logical records) -->
      <Reference URI="#98.200" Type="http://www.w3.org/2000/09/xmldsig#Manifest">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>60NvZvt dTB+7UnLLp/H24p7h4bs=</DigestValue>
      </Reference>

      <!-- Reference to Audit Log Field since we want it included in the signature -->
      <Reference URI="#98.998" Type="AuditLog">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>60NvZvt dTB+7UnLLp/H24p7h4bs=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
juS5RhJ884qoFR8f1VXd/rbrSDVGn40CapgB7qeQiT+rr0NekEQ6BHhUA8dT3+BC
TBUQIOdBjlmL9lwzENXvS83zRECjzXbMRTUtVZiPZG2pgKPnL2YU3A9645UCjTXU
+jgFumv7k78hieAGDzNci+PQ9KRmm//icT7JaYztgt4=
    </SignatureValue>
    <KeyInfo>
      <X509Data>

```

```

        <X509IssuerSerial>
          <X509IssuerName>CN=Test RSA CA,0=Baltimore Technologies\,
            Ltd.,ST=Dublin,C=IE</X509IssuerName>
          <X509SerialNumber>970849928</X509SerialNumber>
        </X509IssuerSerial>
      </X509Data>
    </KeyInfo>
  <Object>
    <SignatureProperties>

      <SignatureProperty Id="BindingAttributes" Target=
"#368eef11-06a2-456c-ac91-5949edaa6722">
        <!-- Binding Attributes -->
        <bi:BindingAttribute bi:Version="1" bi:Method=" 1" icism:Classification="U"
icism:OwnerProducer="USA" >

          <!-- Binder - Person or Service that performs the actual binding -->
          <bi:Binder>
            <!-- This example shows information from an X509 certificate but it
could also be a string, a universally unique identifier
(UUID), or we could define others -->
            <X509IssuerSerial>
              <X509IssuerName>CN=Test RSA CA,0=Baltimore Technologies\,
Ltd.,ST=Dublin,C=IE</X509IssuerName>
              <X509SerialNumber>970849928</X509SerialNumber>
            </X509IssuerSerial>
          </bi:Binder>

          <!-- Requestor - Person or System that requests the binding be created -->
          <bi:Requestor>
            <!-- This example shows information from an X509 certificate but it
could also be a string, a universally unique identifier
(UUID), or we could define others -->
            <X509IssuerSerial>
              <X509IssuerName>CN=Test RSA CA,0=Baltimore Technologies\,
Ltd.,ST=Dublin,C=IE</X509IssuerName>
              <X509SerialNumber>970849928</X509SerialNumber>
            </X509IssuerSerial>
          </bi:Requestor>
        </bi:BindingAttribute>
      </SignatureProperty>
    </SignatureProperties>
  </Object>
</Object>
</Signature>
</itl:ExampleUserDefinedFields>

- <!-- 98.900 IA Audit Log Field (ALF) -->
  <!-- Events should be added in the order they occurred so they can be reversed
successfully -->
  <AuditLog id="98.900">
    <Event Field="1.003" Type="Modification" EventReason="Additional Images were
acquired" Agent="zsimonetti">
      <OldValue <!-- Old value of the attribute goes here (if the field is an
attribute) -->>

```

```
<!-- Old value of the element goes here (if the field is an element)-->
    </OldValue>
  </Event>
</AuditLog>
<!-- 98.901 IA Audit Revision Number (ARN) -->
<AuditRevisionNumber id="98.901" Revision="1">
  </AuditRevisionNumber>
</itl:InformationAssuranceRecord>
```

Appendix D Binding Information Specification (For Traditional Encoding)

This Appendix describes conventions for assuring the relationship between one or more distinct data assets. Note that for the purposes of this Appendix "Distinct Data Assets" Refers to any data that can be expressed as a byte array, such as the logical records of an ITL file.

1. Introduction

This appendix describes a convention for providing integrity and authenticity to a given relationship between one or more distinct data assets via a process known as Cryptographic Binding (CB), which will be referred to simply as Binding. Binding produces a Binding Information (BI) which can then be used to verify the relationship at a later date.

The integrity and authenticity of a cryptographically bound relationships is verified by a Cryptographic Binding Validation (CV) process that uses the BI. This process is referred to simply as Validation.

1.1 Rationale

This section provides a simple example to illustrate the motivation for BI. Consider the scenario where person X authors several document portfolios, each of which is composed of many distinct documents, and X wishes to pass these portfolios to person Y. If X is required to pass one document separately (perhaps one at a time due to file size concerns, or through different media for security considerations) the relationship among documents (namely which documents belong to which portfolio) could be confused or maliciously tampered with (without tampering with the documents themselves) unless an additional explicit description of the relationships (e.g. a BI) is also passed.

Person X would pass each document and the BI for each portfolio to Y. Y would assemble the portfolios as each document is received, and upon completion of a portfolio, use the BI to validate that the correct documents were put into the correct portfolios, and that none of the data assets were tampered with or modified, implying that the portfolios themselves have integrity and authenticity.

1.2. Terminology

In this document, the key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL are to be interpreted as described in [STDWORDS].

2. BI Components

BI is an explicit representation of a relationship between distinct data assets. It MUST include Binding Attributes (BA) and Data Asset List (DAL). It MAY accommodate contents with varying levels of protection as well.

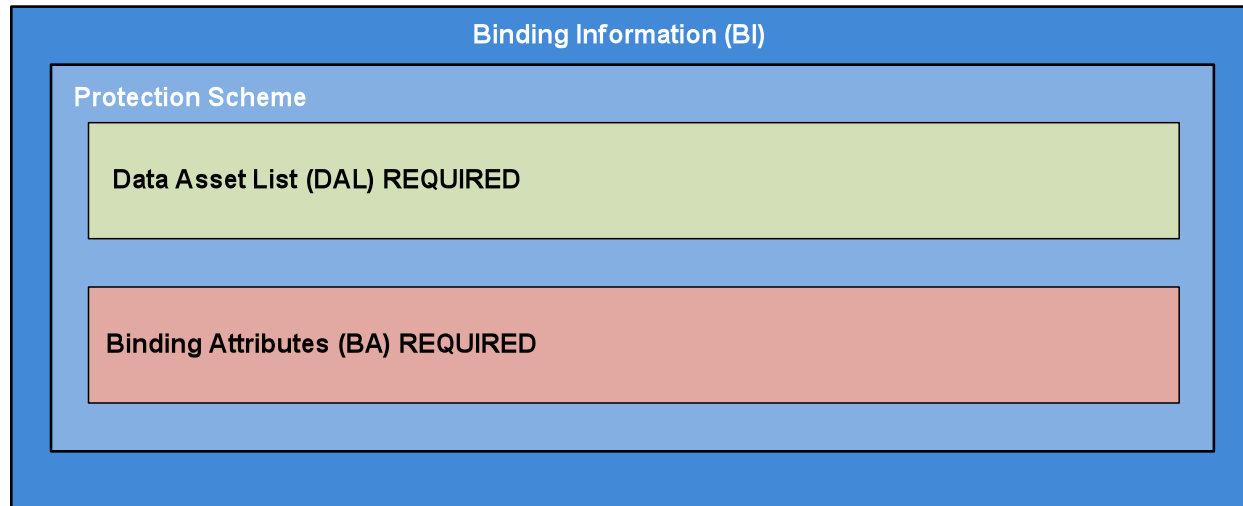


Figure 1. Generalized View of Binding Information (BI)

Figure 1 shows the general order and placement of the components of a BI. Exact syntax and placement depends on the encoding standard and method used.

2.1. Binding Attributes (BA)

Binding Attributes (BA) provide additional information that describes the relationship represented by the BI, as well as information describing the BI itself. For example: the Binding Version and Binding Method are REQUIRED for correct parsing of the BI. The Bind ID, Binder ID, and Requestor ID are OPTIONAL but RECOMMENDED for audit and provenance reasons. The Security Label is OPTIONAL and SHOULD be used when security information about the relationship between the data assets is available.

2.1.1. Binding Version (REQUIRED)

Binding Version is a Binding Attribute expressed as an integer and used to denote the version of the BI specification that was used to create a Binding Information. This is important for interoperability reasons as it allows a validator to correctly parse multiple versions of BIs.

2.1.2. Binding Method (REQUIRED)

Binding Method is a REQUIRED Binding Attribute expressed as integer used to denote the methodology used to establish the relationship between assets as well as the method by which the data assets are included in the DAL. There are three defined methods:

1. Digitally Signed Digest of Data Asset List
2. Encrypted Digest of Data Asset List
3. Hash-based Message Authentication Code (HMAC) of Data Asset List

Other cryptographic methods not explored in this document that use encryption and decryption may be appropriate for creating a BI. For example, the asset (or its hash) can be encrypted so that the validator must prove that it can decrypt the encrypted value correctly for the BI to have

integrity. Authenticity is assured by appropriately managing the encryption key. This is slightly different than using a signature because the signature "sign" and signature "validate" operations are not required to use encryption. Note that certain signature algorithms such as RSA do encrypt the hash of the signed message; but, other algorithms such as the Digital Signature Algorithm (DSA) do not. Another approach is to use the residue or last n bits) of the encryption of an asset.

2.1.3. Bind ID (RECOMMENDED)

Bind ID is a RECOMMENDED unique identifier for the Binding Information itself. It MAY be used by an enterprise to uniquely identify each BI for auditing or tracking purposes.

2.1.4. Binder ID (RECOMMENDED)

The Binder ID SHOULD identify the instance of the software or service creating the BI. This is especially useful within a large scale enterprise or a load balanced environment since it allows for the identification of the particular software or service that created the BI if it was given a unique identifier.

2.1.5. Requestor ID (RECOMMENDED)

The Requestor ID SHOULD identify the entity requesting the BI be created. This is useful for auditing, authoring and tracking purposes. In most cases this should be a unique identifier for a user, such as a name or certificate.

2.1.6. Security Label (OPTIONAL)

The Security Label MUST describe the sensitivity level of the BI.

2.2 Data Asset List (DAL) (REQUIRED)

The Data Asset List (DAL) MUST be a list containing a representation of each data asset in the relationship. That representation SHOULD be a unique identifier and either the entire data asset (depending on size restrictions) or a trustworthy representation such as a hash of the data asset. Using a unique identifier and hash of the data asset is RECOMMENDED.

3. General Requirements of Cryptographic Binding and Validation Systems

Because producing a binding requires both computational resources and the requester's time there may be tradeoffs associated with who should bind and when the binding should occur. In order to make the integrity and authenticity information available to other applications, discovery services or other users as soon as possible, the binding is expected to occur in a timely fashion. This is not always practical in tactical environments where authorized personnel may be fully engaged in a critical activity (war-fighting) or when bandwidth, storage capacity, or other computation resources are simply not available.

Enterprises SHOULD establish policy that addresses who and when to create a binding. For example, radio links may be the only available communication medium at the edge or front line battle space; the speed and capacity of those links may be too slow to request and complete a binding efficiently. Here, binding by the enterprise, perhaps at the platoon level, makes more sense. A local enclave could aggregate binding requests from the field and provide binding enterprise services locally. Best Practice guidance SHOULD be to implement cryptographic binding as close to the user as feasible without compromising the users primary concern.

3.1. General Requirements of Cryptographic Binding (CB)

Regardless of the Binding Method chosen (Methods 1 through 3 are described in 2.1.2), the information needed to apply the protection scheme (i.e. private key, shared secret, etc), the data assets, and the information to create the Binding Attribute are REQUIRED.

The general process involves creating BA and processing the data assets to create the DAL. Once the BA and the DAL are created, the protection scheme is applied.

3.2. General Requirements of Cryptographic Binding Validation (CV)

A successful Validation proves that the relationship between the data assets is authentic and has integrity and that the assets themselves have not been modified (or tampered with) since the BI was created. Validation of a BI MAY occur in three ways: Binding Only, Subset, and Full.

For performance reasons, it is RECOMMENDED that comparisons occur as data assets are received so that failure of one data asset MAY terminate the processing of subsequent data assets if bandwidth is a concern. There are cases however where it is preferential to compare all of the data assets and return a list of the individual results.

3.2.1. Binding Only

Binding Only validation MUST verify that the protection scheme used to protect the BA and DAL is valid. No data asset validation occurs during a Binding Only validation. For example, the Digitally Signed Digest of DAL binding method MUST verify only the signature and Shared Secret and the Digest of DAL binding method MUST verify only the Shared Secret.

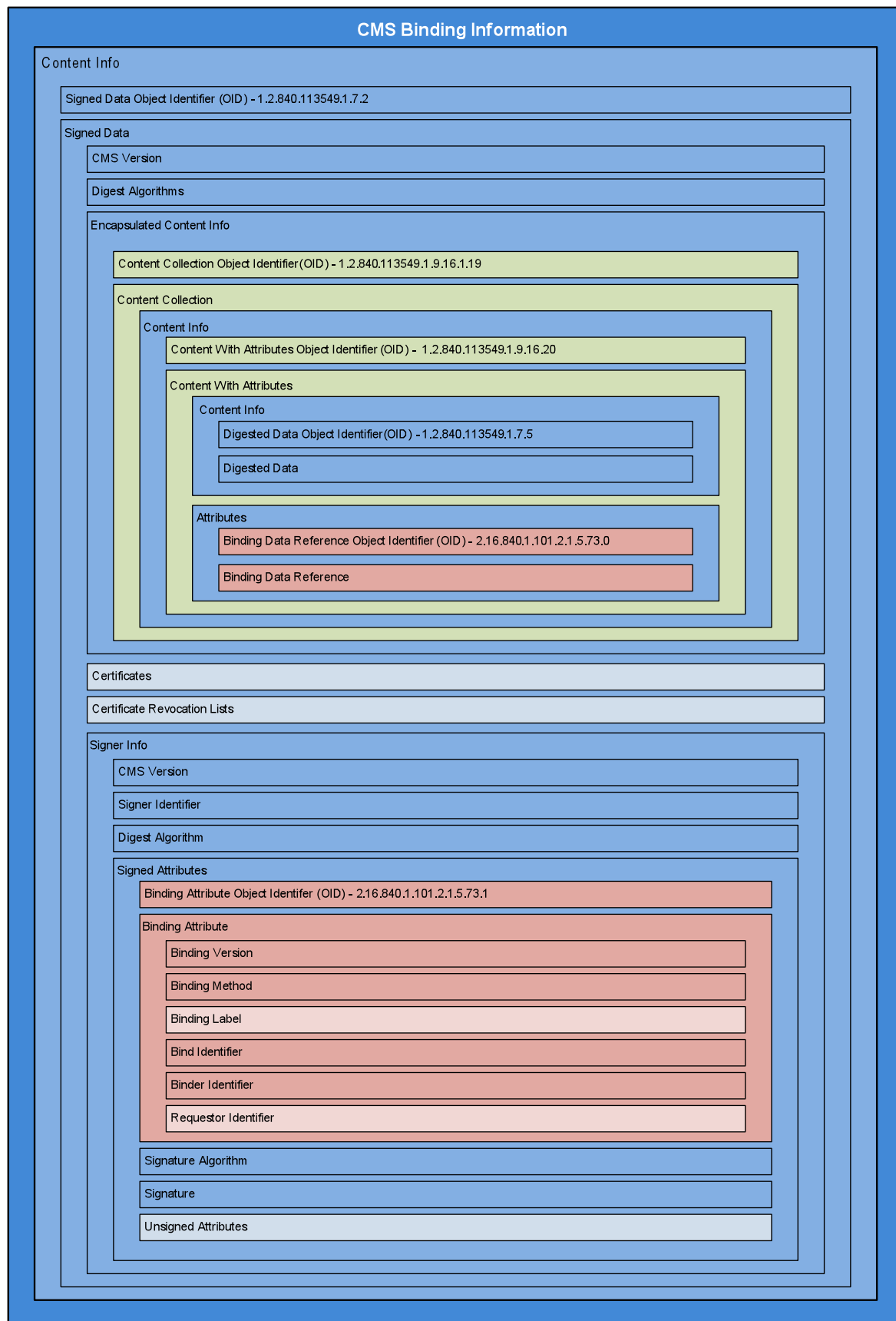
3.2.2. Subset

A subset validation MUST perform a Binding Only validation prior to comparing the data assets requested. The comparison of at least one data asset is REQUIRED for a subset validation. The result of a subset validation SHALL be the result of the Binding Only validation and the results of the comparison of each data asset (i.e. Binding Only Result + Data Asset 1 Comparison + Data Asset 2 Comparison...). Data assets not found in the BI SHOULD be considered a failed comparison for the purposes of the overall result, though providing an explanation as well is RECOMMENDED.

3.2.3. Full

A full validation MUST perform a Binding Only validation prior to comparing the data assets requested. The comparison of all data asset is REQUIRED for a full validation. The result of a full validation SHALL be the result of the Binding Only validation, the results of the comparison of each data asset, and whether or not every data item in the BI was validated (i.e. Binding Only Result + Data Asset 1 Comparison + Data Asset 2 Comparison... + Every data asset compared). Data assets not found in the BI SHOULD be considered a failed comparison for the purposes of the overall result, though providing an explanation as well is RECOMMENDED.

4. Sample Binding Information (BI) For Method 1 using Cryptographic Message



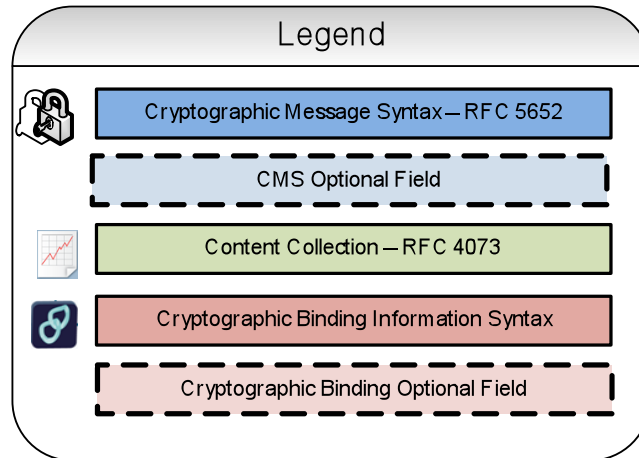


Figure 2. Cryptographic Message Syntax Binding Information (BI) Diagram

Figure 2 shows the order and placement of the components of a BI utilizing [CMS] and [CNTCLLTN].

The following is a textual description of the same Binding Information (BI) and its source documentation.

```

ContentInfo (CMS2004)
contentType ::= SignedData (CMS2004)
content
  version ::= CMSVersion (CMS2004)
  digestAlgorithms ::= DigestAlgorithmIdentifiers (CMS2004)
  encapContentInfo ::= EncapsulatedContentInfo (CMS2004)
  eContentType ::= (ContentCollection)
  eContent
    SEQUENCE OF ContentInfo
    contentType ::= ContentWithAttributes (ContentCollection)
    content
      content ::= ContentInfo
      contentType ::= DigestedData (CMS2004)
      content
        version ::= CMSVersion
        digestAlgorithm ::= DigestAlgorithmIdentifier (CMS2004)
        encapContentInfo ::= EncapsulatedContentInfo
        eContentType ::= id-Data (CMS2004)
        eContent (Not Used)
        digest ::= Digest (CMS2004)
        attrs ::= SEQUENCE OF Attribute (CMS2004)
        reference ::= BindingDataReference (CBInfoSyntax)
        ...
      certificates ::= CertificateSet (CMS2004) (Optional)
      crls (Not Used)
      signerInfos ::= SignerInfos (CMS2004) ::= SET OF SignerInfo (CMS2004)
      version ::= CMSVersion
      sid ::= SignerIdentifier (CMS2004)
      digestAlgorithm ::= DigestAlgorithmIdentifier
      signedAttrs ::= SignedAttributes (CMS2004)
      bindingAttr ::= BindingAttr (CBBindingInfoSyntax)
      signingTime ::= SigningTime
      signatureAlgorithm ::= SignatureAlgorithmIdentifier (CMS2004)
      signature ::= SignatureValue (CMS2004)
      unsignedAttrs (Not Used)

```

4.1. Supporting Types

4.1.1. Binding Data Reference Type

The BindingDataReference type is used to uniquely identify the external data asset that the Digested Data represents. The syntax supports multiple methods of identifying the external data asset. For example, the BindingDataReference could contain a PrintableString with the value of the asset's file name and extension.

The following object identifier names the BindingDataReference type:

```
id-cryptographicBindingDataReference OBJECT IDENTIFIER ::=
    {joint-iso-ccitt(2) country(16) us(840) organization(1) gov(101) dod(2)
    infosec(1) attributes(5) cryptographicBinding(73) 0 }
```

The BindingDataReference has the following syntax:

```
BindingDataReference ::= CHOICE {
    uuid [0]UUID,
    printableString PrintableString (SIZE(1..MAX)),
    octetString [1]OCTET STRING (SIZE(1..MAX)),
    attribute BindingDataReferenceAttribute
}
```

The BindingDataReference type has three defined choices and a fourth choice, the BindingDataReferenceAttribute type, that allows for more flexibility in uniquely identifying external data assets.

The BindingDataReferenceAttribute has the following syntax:

```
BindingDataReferenceAttribute ::= Attribute
```

4.1.2. Entity Identifier type

The EntityIdentifier type is used to uniquely identify an entity, such as a person, business, system, government agency, etc.

The EntityIdentifier type has the following syntax:

```
EntityIdentifier ::= CHOICE {
    issuerSerial [0]IssuerSerial,
    uuid [1]UUID,
    printableString PrintableString (SIZE(1..MAX)),
    attribute EntityIdentifierAttribute
}
```

The EntityIdentifier type has three defined choices and a fourth choice, the EntityIdentifierAttribute type, that allows for more flexibility in uniquely identifying external data assets.

The EntityIdentifierAttribute has the following syntax:

```
EntityIdentifierAttribute ::= Attribute
```

4.1.3. Binding Attribute Type (REQUIRED)

The BindingAttribute type is used to consolidate important processing and audit data about the BI. A BindingAttribute is expected to be protected. In this case the BindingAttribute is placed as a SignedAttribute of each SignerInfo. It could also be placed inside of the EncapsulatedData type however that reduces the flexibility of the binding if countersignatures are utilized.

The following object identifier names the BindingAttribute type:

```
id-cryptographicBindingDataReference OBJECT IDENTIFIER ::=
    {joint-iso-ccitt(2) country(16) us(840) organization(1) gov(101)
    dod(2) infosec(1) attributes(5) cryptographicBinding(73) 1 }
```

The BindingAttribute type has the following syntax:

```
BindingAttribute ::= SEQUENCE {  
    version BindingVersion,  
    method BindingMethod,  
    securityLabel SecurityAttribute OPTIONAL,  
    bindID BindingIdentifier,  
    binderID EntityIdentifier,  
    requestorID EntityIdentifier OPTIONAL  
}
```

4.1.3.1. Binding Version (REQUIRED)

Binding Version is defined in the BindingAttribute type as:

```
version BindingVersion
```

The BindingVersion type has the following syntax:

```
BindingVersion ::= INTEGER
```

4.1.3.2. Binding Method (REQUIRED)

Binding Method is defined in the BindingAttribute type as:

```
method BindingMethod
```

The BindingMethod type has the following syntax:

```
BindingMethod ::= INTEGER
```

4.1.3.3. Bind ID (REQUIRED)

Bind ID is defined in the BindingAttribute type as:

```
bindID BindingIdentifier
```

The BindingIdentifier type has the following syntax:

```
BindingIdentifier ::= UUID
```

In this particular example and most other cases, it will be REQUIRED. However it is not necessarily vital for processing the BI, so the overall specification lists it as RECOMMENDED.

4.1.3.4. Binder ID (REQUIRED)

Binder ID is defined in the BindingAttribute type as:

```
binderID EntityIdentifier
```

It utilizes the EntityIdentifier type to uniquely identify the entity that created the binding.

4.1.3.5. Requestor ID (RECOMMENDED)

Requestor ID is defined in the BindingAttribute type as:

```
requestorID EntityIdentifier OPTIONAL
```

In this example it is marked OPTIONAL, however it is generally RECOMMENDED that it be filled for audit purposes.

4.1.3.6. Security Label (OPTIONAL)

Security Label is defined in the BindingAttribute type as:

```
securityLabel SecurityAttribute OPTIONAL
```

The SecurityAttribute type has the following syntax:

```
SecurityAttribute ::= Attribute
```

The SecurityAttribute has a type of Attribute so that many different security methodologies can be supported without the need to redefine the BindingAttribute type for each specific methodology.

4.2. Data Asset List (DAL) (REQUIRED)

The Data Asset List using [CMS] and [CNTCLLTN] SHOULD be comprised of a ContentCollection containing a collection of ContentWithAttributes as described in 1.2 of [CNTCLLTN]. Each ContentWithAttributes SHOULD contain a DigestedData as the content, and a BindingDataReference as an attribute. Additional attributes SHOULD be supported.

4.3. Protection Scheme

This example utilizes digital signatures and the SignedData type described in [CMS] to protect the BA and DAL. This example could be modified for other methods by utilizing the other CMS protecting content types.

4.4. ASN.1 Modules

The ASN.1 module contained in this appendix defines the structures that are needed to implement this specification. It is expected to be used in conjunction with the ASN.1 modules in [CMS], [CNTCLLTN] and [COMPRESS].

```
CryptographicBindingInformationSyntax { joint-iso-ccitt(2) country(16)
    us(840) organization(1) gov(101) dod(2) infosec(1) attributes(5)
    cryptographicBinding(73) }
```

```
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
```

```
-- EXPORTS ALL
```

```
IMPORTS
```

```
-- Imports from RFC 3852
Attribute
    FROM CryptographicMessageSyntax2004
    { 1 2 840 113549 1 9 16 0 24 }
```

```
-- Imports from RFC 3281
IssuerSerial
    FROM PKIXAttributeCertificate
    { 1 3 6 1 5 5 7 0 12 };
```

```
id-cryptographicBindingInfo OBJECT IDENTIFIER ::= { joint-iso-ccitt(2)
    country(16) us(840) organization(1) gov(101)
    dod(2) infosec(1) attributes(5)
    cryptographicBinding(73) }
```

```
id-cryptographicBindingDataReference OBJECT IDENTIFIER ::=
{ joint-iso-ccitt(2) country(16) us(840)
    organization(1) gov(101) dod(2) infosec(1)
    attributes(5) cryptographicBinding(73) 0 }
```

```
id-cryptographicBindingAttribute OBJECT IDENTIFIER ::= { joint-iso-ccitt(2)
    country(16) us(840) organization(1) gov(101)
    dod(2) infosec(1) attributes(5)
    cryptographicBinding(73) 1 }
```

```
BindingAttribute ::= SEQUENCE {
    version BindingVersion,
    method BindingMethod,
    securityLabel SecurityAttribute OPTIONAL,
```

```

        bindID BindingIdentifier,
        binderID EntityIdentifier,
        requestorID EntityIdentifier OPTIONAL
    }

BindingDataReference ::= CHOICE {
    uuid [0]UUID,
    printableString PrintableString (SIZE(1..MAX)),
    octetString [1]OCTET STRING (SIZE(1..MAX)),
    attribute BindingDataReferenceAttribute
}

EntityIdentifier ::= CHOICE {
    issuerSerial [0]IssuerSerial,
    uuid [1]UUID,
    printableString PrintableString (SIZE(1..MAX)),
    attribute EntityIdentifierAttribute
}

BindingIdentifier ::= UUID
BindingVersion ::= INTEGER
BindingMethod ::= INTEGER
BindingDataReferenceAttribute ::= Attribute
EntityIdentifierAttribute ::= Attribute
SecurityAttribute ::= Attribute
UUID ::= OCTET STRING (SIZE(16))

END

```

5. Security Considerations

5.1. Algorithm and Key Strength Guidance

This section looks at expectations for how well hash algorithms, cryptographic algorithms, and keys protect the integrity and authenticity of a BI. Each has an impact on the cryptographic binding and validation system when considering the strength-of- mechanism in specific development and execution environments. The material in this section is organized to address the cryptographic nature of the five cryptographic methods envisioned for use when creating and validating a binding. The material in this section is drawn from [NSP80057] and [NSP800107]. These documents offer the following working definitions needed to describe strength-of-mechanism:

1. Collision - Two different known messages that have the same hash value (message digest).
2. Entropy - A measure of the disorder, randomness, or variability in a closed system. The entropy of X is a mathematical measure of the amount of information provided by an observation of X (i.e. a key can have 128 bits of entropy).
3. Security strength - A number associated with the amount of work, the number of operations or work factor that is required to break a property of a cryptographic algorithm or system; security strength is specified in bits.
4. Work factor - A number of executions of an algorithm by an adversary that is required to break some property of the algorithm. For example, for SHA-256, a work factor of 2128 executions of the algorithm is required by an adversary to find a collision.

5.2. Hash Algorithm Strength

A hash or message digest produced by an approved hash algorithm has one or more of the following properties:

1. Collision resistance - It is computationally infeasible to find two different inputs to the hash function that have the same hash value. The amount of collision resistance provided by a hash-function cannot exceed half the length of the hash value produced by a given hash function. Collision resistance is measured in bits. For example, SHA-256 produces a (full length) hash value of 256 bits; SHA-256 cannot provide more than 128 bits of collision resistance.
2. Preimage resistance - It is computationally infeasible to find a message that hashes to a given value. This property is also called the one-wayness property. Preimage resistance is measured by the amount of work that would be needed to find a preimage for a hash function. The amount of preimage resistance provided by a hash-function cannot exceed the length of the hash value produced by a given hash function. For example, SHA-256 cannot provide more than 256-bits of preimage resistance; this means that a work factor of 2256 operations will likely find a preimage of a (full length) SHA-256 hash value.
3. Second preimage resistance - It is computationally infeasible to find another different message that has the same hash value as the first message. Second preimage resistance is measured by the amount of work that would be needed to find a second preimage for a hash function. The amount of second preimage resistance provided by a hash-function cannot exceed the length of the hash value produced by a given hash function. For example, SHA-256 cannot provide more than 256-bits of second preimage resistance.

Hash Algorithm	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Collision Resistance	<80	112	128	192	256
Preimage Resistance	160	224	256	384	512
Second Preimage Resistance	160 - K (105-160)	224 - K (201-224)	256 - K (201-256)	384	512 - K (394-512)

Table 1. Strength (In Bits) of Hash Algorithms

Table 1 lists the strength in bits of the hash algorithms approved for use in U.S. Government systems. The following two notes apply to Table 1:

1. The second preimage resistance strengths of the hash functions depend not only on the functions themselves, but on the sizes of the messages that the hash functions process. As shown in the third row of Table 1, the second preimage resistance strength is $(L - K)$, where L is the output block size of the hash algorithm. For instance, in SHA-256 $L = 256$, and K is a function of the input block size. The strength in bits is given by $2k = \text{message length/block size}$. The values in parentheses show the limits for the block sizes of each function. As discussed in detail in the appendix of [35], the second preimage resistance of SHA-384 does not depend on the message length because the Second Preimage Attack requires the work of more than 2384, and to break the second preimage resistance of SHA-384, the required work is only 2384.
2. The preimage resistance is stronger than its collision resistance strength. Therefore, if a hash algorithm satisfies a collision resistance requirement it also satisfies any preimage resistance requirement. Similarly, if K is not greater than $L/2$ (half of the output block size in bits) or $(L - K) = L/2$, then the second preimage resistance strength is equal to or greater than its collision resistance strength. In this case, if the hash function satisfies a collision resistance requirement then it also satisfies any second preimage resistance requirement. A value of K that is less than or equal to $L/2$ is very common, in practice. For example, if a hash function in a

digital signature application satisfies the collision resistance requirement, and the messages hashed by the application is not longer than $2(L/2)$ input blocks of the hash function in length, then it also satisfies the second preimage resistance requirement.

5.3. Hashes for Method 1

The security strength of a hash function for digital signatures as used in cryptographic binding methods 1 is its collision resistance strength. However, when using the randomized hashing technique as specified in [NSP800106], the security strength of the hash algorithm is the second preimage resistance strength, not the collision resistance strength. It is observed that when using XML DSIG or the equivalent CMS signature, the value of K as described in note 2 above will typically be much less than $L/2$ therefore if a hash algorithm satisfies a collision resistance requirement it also will likely satisfy any preimage or second preimage requirement.

5.4. Hashes for Method 3

The security strength of a hash function for HMAC applications as used in method 3 is its preimage resistance strength and the strength of the authenticated shared secret key. The key must contain at least $L/2$ bytes (i.e., $8 * L/2$ bits) of entropy, so that at least $(8 * L/2)$ bits of security are provided, where L is the length (in bytes) of the hash function output. Approved key generation methods include the generation of random bits using a random bit generator as specified in [NSP80090], and the use of an approved key establishment method (e.g., a method specified in [NSP80056]).

5.5 Encryption, Signature and Key Exchange Algorithm Strength

Encryption, signature, and key exchange algorithms provide different strengths depending on the algorithm, its modes of operation, and the key size used. Two algorithms are considered of comparable strength for a given key size if they require approximately the same resources to break. Given a few plaintext blocks and a corresponding cipher, an algorithm that provides X bits of security would, on average, take 2^{X-1} operations for a full exhaustive attack. In general, this is oversimplified as determining the strength of an algorithm relies on other factors. A more detailed description of algorithm strength is found in [NSP80057] from which Table 2 is abstracted for use in creating and validating cryptographic bindings.

The Bits are the number of bits of security provided by the algorithms and key sizes in a particular row. The bits of security are not necessarily the same as the key sizes for the algorithms in the other columns, due to attacks on those algorithms that provide computational advantages. 2TDEA and 3TDEA are specified in the recommendations for the [NSP80067]. For the RSA column, k is the key size; and, for the ECDSA column, f is the range of key sizes.

Bits of Security	Symmetric Key	RSA	ECDSA
80	2TDEA	$k = 1024$	$f = 160-223$
112	3TDEA	$k = 2048$	$f = 224-255$
128	AES-128	$k = 3072$	$f = 256-383$
192	AES-192	$k = 7680$	$f = 384-511$
256	AES-256	$k = 15360$	$f = 512+$

Table 2. Comparable Cryptographic Algorithm Strength

5.6 Comparable Hash Function Strength

For hash algorithms, the size of the hash function will be determined by the algorithm or scheme in which the hash is used. For example, the appropriate hash algorithm for a digital signature algorithm depends upon the chosen key and parameter size, and the security strength to be provided by the digital signature. To further illustrate this concept, Table 3 indicates the hash size with comparable bit strength for the listed parameter and key sizes for digital signatures, HMAC, key derivation functions, and random number generation as they might be used for cryptographic binding and validation functions using methods 1 through 3.

Bits of Security	Signature and Hash Only Applications	HMAC	Key Derivation	Random Number Generation
80	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,
112	SHA-224, SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,
128	SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512,
192	SHA-384, SHA-512,	SHA-224, SHA-256, SHA-384, SHA-512,	SHA-224, SHA-256, SHA-384, SHA-512,	SHA-224, SHA-256, SHA-384, SHA-512,
256	SHA-512,	SHA-256, SHA-384, SHA-512,	SHA-256, SHA-384, SHA-512,	SHA-256, SHA-384, SHA-512,

Table 3. Comparable Hash Algorithm Strength

This table implies a rather subtle performance enhancement possibility: The use of SHA-1 with HMAC can produce 112 bits of security as shown in the 112 row under the HMAC column. To obtain the same 112 bits of security with a digital signature requires SHA 224. This has implications for recommendations through the year 2030 as shown in Table 4 which recommends algorithm and key sizes for cryptographic binding.

5.7 Recommended Algorithms and Key Sizes for Cryptographic Bindings

The different cryptographic binding methods require the use of several different cryptographic algorithms and hash functions. Method 1 uses hashes and signatures, method 2 uses symmetric algorithms and method 3 uses symmetric algorithms and hashes. Methods 2 and 3 will likely use some mechanism to generate and protect the exchanged authenticated key which is likely generated using a random number generator (optionally based on [NSP80090]) and a hash algorithm in concert with an appropriate authenticated shared secret communications protocol. Some algorithms can

be used to perform the same service more efficiently because of their design. For example AES has been designed to be more efficient than TDEA. In many cases, a variety of key sizes may be available for an algorithm. For some of the algorithms (e.g., public key algorithms, such as RSA), the use of larger key sizes than are required may impact operations, e.g., larger keys may take longer to generate or longer to process the data. However, the use of key sizes that are too small may not provide adequate security.

Table 4 abstracted from [NSP80057] provides recommendations that may be used to select an appropriate suite of algorithms and key sizes for Federal Government unclassified applications. For the protection of Federal Government unclassified but sensitive information, [NSP80057] states that, "A minimum of eighty bits of security shall be provided until the year 2010. Between 2011 and 2030, a minimum of 112 bits of security shall be provided. Thereafter, at least 128 bits of security shall be provided."

Table 4 shows these time periods in column 1 and indicates the estimated time periods during which data protected by specific cryptographic algorithms remains secure. (i.e., the algorithm security lifetimes). Column 2 indicates the equivalent minimum key size when RSA is used for digital signatures in methods 1 or 2 and Column 3 indicates the equivalent minimum key size when ECDSA signatures are used in methods 1 and 2. Column 4 identifies appropriate symmetric key algorithms and key sizes for 2TDEA, 3TDEA and AES.

Algorithm Lifetime	Method 1		Method 2	Method 3
	RSA	ECDSA	Symmetric Key	HMAC
Through 2010 (Minimum of 80 bits of strength)	Minimum k = 1024	Minimum f = 160	2TDEA 3TDEA AES-128 AES-192 AES-256	SHA-1
Through 2030 (Minimum of 112 bits of strength)	Minimum k = 2048	Minimum f = 224	3TDEA AES-128 AES-192 AES-256	SHA-224
Through 2030 (Minimum of 128 bits of strength)	Minimum k = 3072	Minimum f = 256	AES-128 AES-192 AES-256	SHA-56

Table 4. Recommended Algorithm and Key Size for Cryptographic Bindings

While no data is specified for HMAC (column 5) in [NSP80057], using the HMAC Publication [7] suggests that the size of the key, K, shall be equal to or greater than $L/2$, where L is the size of the hash function output. Note that keys greater than L bytes do not significantly increase the function strength. Applications that use keys longer than B-bytes, where B is the block size of the input to the hash function, shall first hash the key using H and then use the resultant L-byte string as the HMAC key, K. Keys shall be chosen at random using an approved key generation method and shall be changed periodically. Note that the keys should be protected in a manner that is consistent with the value of the data that is to be protected (i.e., the binding that is authenticated using the HMAC function).

The successful verification of a MAC does not completely guarantee that the accompanying binding is authentic: there is a chance that a source with no knowledge of the key can present a purported MAC on the plaintext message that will pass the verification procedure. For example, an arbitrary purported MAC of t bits on an arbitrary plaintext message may be successfully verified with an expected probability of $(1/2)^t$. This limitation is inherent in any MAC algorithm. With these ideas in mind, appropriate Hash algorithms are suggested in column 5 of

Table 4 for HMAC.

These recommended algorithms and key sizes for cryptographic bindings lend themselves to many different possible implementations based on methods, algorithms and key size options. The prototype implementations described in Section 2.1 represent design points in this space:

- The XML DSIG implementation [5] meets the conditions of Row 1 where the algorithm lifetime is useful through 2010. It uses RSA for Method 1 with a key size of 1024. It uses SHA - 1 providing a minimum of 80 bits of strength;
- The ASN.1 CMS implementation [6] is very flexible meeting or exceeding the conditions of all three Rows. It optionally uses RSA or ECDSA for Method 1 with RSA key sizes of 1024 or 2048 and ECDSA key sizes with NIST P256 or NIST P384 named curves. Each option works with SHA - 1, 256, 384, or 512 depending on the X.509 PKI certificate used.

Appendix E References

6.1. Normative References

- [ASN1] CCITT. Recommendation X.680: ISO/IEC 8824 (All parts)
ITU-T Recommendation X.680-series, *Information Technology*
- *Abstract Syntax Notation One (ASN.1)* 2008.
- [CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC
5652, September 2009.
- [COMPRESS] Gutmann, P., "Compressed Data Content Type for
Cryptographic Message Syntax (CMS)", RFC 3274, June 2002.
- [CNTCLLTN] Housley, R., "Protecting Multiple Contents wit the
Cryptographic Message Syntax (CMS)", May 2005.
- [STDWORDS] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2. Informative References

- [MIME] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
Extensions (MIME) Part One: Format of Internet Message
Bodies", RFC 2045, November 1996.
- [MSG] Ramsdell, B., "Secure/Multipurpose Internet Mail
Extensions (S/MIME) Version 3.1 Message Specification",
RFC 3851, July 2004.
- [NSP80056] NIST Special Publication 800-56 B, "Recommendation for Pair-Wise
Key Establishment Schemes Using Integer Factorization
Cryptography", August 2009.
- [NSP80057] NIST Special Publication 800-57, Part 1 "Recommendation for Key
Management" March, 2007
- [NSP80067] NIST Special Publication 800-67, "Recommendation for the Triple
Data Encryption Algorithm (TDEA) Block Cipher", Revised 19 May
2008.
- [NSP80090] NIST Special Publication 800-90, "Recommendation for Random
Number Generation Using Deterministic Random Bit Generators
(Revised)", March 2007.
- [NSP800106] NIST Special Publication 800-106, "Randomized Hashing for
Digital Signatures", February 2009
- [NSP800107] NIST Special Publication 800-107, "Recommendation for
Applications Using Approved Hash Algorithms" February 2009.